時間枠制約付き配送計画問題に対する局所探索法の適用について (最適化のための連続と離散数理)

AUTHOR(S):

増田, 友泰; 柳浦, 睦憲; 茨木, 俊秀

CITATION:

増田, 友泰 ...[et al]. 時間枠制約付き配送計画問題に対する局所探索法の適用について (最適化のための連続と離散数理). 数理解析研究所講究録 1999, 1114: 194-205

ISSUE DATE:

1999-11

URL:

http://hdl.handle.net/2433/63384

数理解析研究所講究録
1114巻 1999年 194-205

時間枠制約付き配送計画問題に対する局所探索法の適用について

京都大学 情報学研究科　増田 友泰　MASUDA Tomoyasu
京都大学 情報学研究科　柳浦 睦憲　YAGIURA Mutsunori
京都大学 情報学研究科　茨木 俊秀　IBARAKI Toshihide

1 序論

配送計画問題 (Vehicle Routing Problem, VRP) は, 様々な制約条件の下で複数の車両を用い, 全ての客をちょうど一回ずつ訪問するような経路集合の中で, 距離の総和が最短のものを求める問題である. これは, 代表的な組合せ最適化問題の一つであると同時に, 実用性のある問題で, 郵便・新聞配達, 廃棄物収集, 石油運搬やスクールバスのスケジューリングなどの応用を持つ. この問題はNP困難であることが知られており, 大規模な問題例に対して厳密な最適解を求めることは現実的に極めて困難であると考えられている. そのような問題に対する妥協策として近似解法があり, 配送計画問題に対しても種々の近似解法が提示されている.

本研究では, 配送計画問題の制約条件として, 各車両の容量制約と時間枠制約を取り扱う. 容量制約とは, 一台の車両が訪問する客の要求量の総和が, 一台の車両で処理出来る量を越えないというものである. 時間枠制約とは, 客が指定した時間枠内にサービスを開始しなければならないというものである. いずれの制約も, 容量制約のみを課した問題は分割問題を, 時間枠制約のみを課した問題はスケジューリング問題を特殊な場合として含むことから [1], 一方の制約のみを課した問題に対する実行可能解の存在の判定がすでにNP完全である. そのため探索を実行可能解に限定してしまう手法 (例えば [2]) では, 制約が厳しい場合には探索が効率よく行えないと考えられる. また現実問題では, 要求量や時間枠自体があいまいで, 厳密でないことが多い. そこで, これらの制約を考慮制約として扱い, 制約の違反量に応じたペナルティを付加した目的関数を最小化するという方針をとる. 制約を必ず満たす必要がある場合でも, ペナルティを十分大きくとることである程度対応できるので, このような定式化の方がより汎用性が高いと言える.

ところで, このようなペナルティの処理は, 容量制約については容易であるが, 時間枠制約についてこれを実現するためには, 一つの車両が処理する客の訪問順序が決まったとき, ペナルティが最小となるような各客のサービス時刻を決定する問題を解く必要がある. 本研究では, この問題が動的計画法を用いて効率的に解けることを示す. なお, 提案する動的計画法は, 時間枠制約に対するペナルティ関数が区分線型関数であれば, 不連続や非凸であっても扱える. さらに, この動的計画法を組み込んだ局所探索法を提案し, その効果を計算実験により確認する.

2 問題

本節では, 本研究で扱う時間枠制約付き配送計画問題を定義する. 節点集合 $V = \{0, 1, ..., n\}$ に対する完全有向グラフ $G = (V, E)$ と車両集合 $M = \{1, ..., m\}$ を考える. ここで節点0は “デポ”

と呼ばれる特殊な節点であり, また, 他の節点はサービスを受ける客を表す. 各客 i $(i = 0, 1, ..., n)$ には, サービスの要求量 q_i (ただし $q_0 = 0$), サービス開始時刻に対するペナルティ関数 $p_i(t)$, サービス時間 u_i (ただし $u_0 = 0$) が与えられ, 各車両 k $(k = 1, 2, ..., m)$ には, 処理できる要求量の上限 Q_kとデポを出発する時刻 e_0が与えられる. さらに, 非対称距離行列 (d_{ij}) と非対称移動時間行列 (t_{ij}) が与えられる. ペナルティ関数は, 区分線型関数とする. ただし, 非凸, 不連続であっても良い.

各車両 kの客の訪問順序をσ^kとし, $\boldsymbol{\sigma} = (\sigma^1, ..., \sigma^m)$ とする. ただし, 客の訪問順序を決定する際には以下の制約条件が付く.

- 各車両 k $(k = 1, ..., m)$ はデポを出発し, デポに帰還する.
- 各客はちょうど 1 回だけある車両によりサービスされる.

このとき, 全ルートの距離の総和を $D(\boldsymbol{\sigma})$, 各客のサービス時刻に対するペナルティの総和を $T(\boldsymbol{\sigma})$, 容量超過量の総和を $Q(\boldsymbol{\sigma})$ とすると, 上述の 2 つの制約条件を満たした上で最小化すべき目的関数は以下のようになる.

$$cost(\boldsymbol{\sigma}) = D(\boldsymbol{\sigma}) + T(\boldsymbol{\sigma}) + \alpha Q(\boldsymbol{\sigma}). \tag{1}$$

ただし, αは, 容量制約違反のペナルティに対する重み係数である.

3 局所探索法

まず, 局所探索法が探索の対象とする探索空間について考察する. 客の訪問順序$\boldsymbol{\sigma} = (\sigma^1, ..., \sigma^m)$を決定すると, 目的関数における $D(\boldsymbol{\sigma})$ と $Q(\boldsymbol{\sigma})$ の値は決定される. また, $T(\boldsymbol{\sigma})$ については, 3.2節の動的計画法により, $\boldsymbol{\sigma}$を固定するという条件の下で $T(\boldsymbol{s})$ が最小となるように各客 i のサービス開始時刻 s_iを一意的に決定することができる. 以上より, 探索空間を実行可能解$\boldsymbol{\sigma}$の集合とする.

実行可能解$\boldsymbol{\sigma}$の近傍 $NB(\boldsymbol{\sigma})$ とは, $\boldsymbol{\sigma}$に対して適当な操作を加えて得られる実行可能解の集合のことである. 局所探索法とは, 現在の解$\boldsymbol{\sigma}$の近傍 $NB(\boldsymbol{\sigma})$ 内に$\boldsymbol{\sigma}$より良い解があればそれに置き換える, という操作を反復するものである.

LOCAL SEARCH (LS($\boldsymbol{\sigma}$))

0. 適当な近似解法で初期解を求め, $\boldsymbol{\sigma}$とする. $k = 1$ とする.

k. $\boldsymbol{\sigma}$の近傍 $NB(\boldsymbol{\sigma})$ 内で$\boldsymbol{\sigma}$より良い目的関数値をもつ実行可能解$\boldsymbol{\sigma}'$を探す. そのような$\boldsymbol{\sigma}'$が見つかれば, 解を$\boldsymbol{\sigma} := \boldsymbol{\sigma}'$と更新したのち $k := k + 1$ として k へ. そうでなければ現在の暫定解$\boldsymbol{\sigma}$を返す.

局所探索法においては, 近傍をどのように定義するかによって最終的に得られる解の精度が大きく異なる. したがって個々の問題に応じた効率の良い近傍を定義する必要がある. 以下では, 本研究で用いる 3 つの近傍について述べた後, それらの近傍の探索順序について述べる. 次に, ルートが与えられた時の各客の最適なサービス時刻を決定する動的計画法について述べる. なお, 局所探索法において, 近傍内の解をどのような順序で調べるかには色々考えられるが, これを順序良く行うことで, 動的計画法の 1 回あたりの計算量を大幅に削減できる. よって, 本研究で提案する局所探索法の詳細と, 動的計画法のそのような高速化については, 最後にまとめて説明を行う. また, 暫定解の評価法についても簡単に触れる.

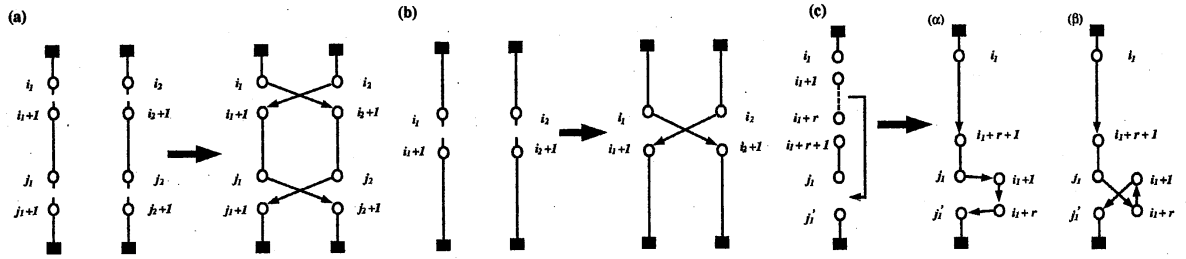

図 1: クロス交換近傍 (a), 2-opt*近傍 (b), およびルート内挿入近傍 (c) の操作の例.
(c) については, パスの順序を変えない場合 (α) と反転する場合 (b) の 2 通りを示す.

3.1 近傍

3.1.1 クロス交換近傍と 2-opt*近傍

異なる 2 つのルート間にまたがる近傍操作としてクロス交換操作と 2-opt*操作がある.

まずクロス交換操作であるが, これは, 異なる 2 つのルートのそれぞれから長さ L^{cross} 以下のパスを選び, それらを互いに交換する操作である (図 1(a)). そして, 現在の解$\boldsymbol{\sigma}$に対してこの操作を加えることによって得られる解集合の全体をクロス交換近傍 $N^{cross}(\boldsymbol{\sigma})$ と呼ぶことにする. 現在の解$\boldsymbol{\sigma} = (\sigma^1, ..., \sigma^m)$ について, 2 つの異なるルートσ^Aとσ^Bに対してクロス交換操作を行って得られる解集合を $N^{cross}(\boldsymbol{\sigma}, A, B)$ とすると,

$$N^{cross}(\boldsymbol{\sigma}) = \bigcup_{A<B} N^{cross}(\boldsymbol{\sigma}, A, B) \tag{2}$$

となる.

2-opt*近傍操作とは, 異なる 2 本のルートそれぞれから 1 本ずつ枝を取り除くことで, 各ルートを前半と後半の 2 つのパスに分け, 後半のパスをルート間で互いに交換する操作である (図 1(b)). 2 つの異なるルートσ^A, σ^Bに対して 2-opt*操作を行って得られる解集合を $N^{2opt^*}(\boldsymbol{\sigma}, A, B)$, 2-opt*近傍の全体を $N^{2opt^*}(\boldsymbol{\sigma})$ と表すと,

$$N^{2opt^*}(\boldsymbol{\sigma}) = \bigcup_{A<B} N^{2opt^*}(\boldsymbol{\sigma}, A, B) \tag{3}$$

となる.

3.1.2 ルート内挿入近傍

同一ルート内での近傍操作としてルート内挿入操作がある. これは, 長さ L^{intra} 以下のパスを同一ルートの他の位置へ挿入する操作 (図 1(c)) で, この操作によって得られる解集合をルート内挿入近傍と呼ぶ. ルートσ^Aに対してルート内挿入操作を行って得られる解集合を $N^{intra}(\boldsymbol{\sigma}, A)$, ルート内挿入近傍の全体を $N^{intra}(\boldsymbol{\sigma})$ と表すと,

$$N^{intra}(\boldsymbol{\sigma}) = \bigcup_{A\in M} N^{intra}(\boldsymbol{\sigma}, A) \tag{4}$$

となる. なお, ルート内挿入近傍においては, 取り除いたパスの順序を反転させて挿入するという操作も加える. この操作を加えても, 3.3.2節で述べる計算方法によって, これを行わない場合とオーダー的にほぼ同等の計算時間でコストの変化量の計算が可能である.

3.1.3 探索順序

本研究の局所探索法における上記で定めた近傍の探索順序は,

1. ルート内挿入近傍探索
2. 2-opt*近傍探索
3. クロス交換近傍探索

である. さらに, 各探索は, その近傍内での解の改善が起こらなくなるまで行う. すなわち, ある探索によって解の改善が起こると, 繰り返し同じ近傍を探索するのである. また, クロス近傍探索を終えるとルート内挿入近傍探索に戻る. このループは, 上記の 3 つの近傍全てにおいて解の改善が起こらなくなるまで続ける.

3.2 与えられたルートの最適サービス時刻の決定法

あるルートの客の訪問順序が与えられた時に, そのルートの時間枠に関するペナルティの総和が最小となるように, 各客のサービス時刻を決定する問題を考える.

車両 kが h 番目に処理する客が i であるとき, $\sigma^k(h)=i$ と記す. ルートσ^k内の客数を n_k, 時間枠制約のペナルティを表す区分線型関数の区分の数を, 車両 kが処理する全ての客について和をとったものをδ_kとする. また, 便宜上, $\sigma^k(0)=0, \sigma^k(n_k+1)=0$ とおく.

本節では, このサービス時刻決定問題が動的計画法を用いて $O(n_k\delta_k)$ 時間で計算出来ることを示す. この問題は, 各客のペナルティ関数が凸関数ならば, 凸計画問題として定式化でき, 汎用的な解法で解ける. 一方, 本研究で提案する動的計画法では, ペナルティ関数が非凸や不連続であっても効率良く解ける.

3.2.1 アルゴリズム

ルートσ^kの最適サービス時刻の決定を考える. 関数 $f_h^k(t)$ を, ルートσ^kにおける h 番目の客のサービス開始時刻が t 以前であるときのペナルティの最小値と定義し, これを以後, 前向きペナルティ最小関数と呼ぶ. また, 便宜上, $p_h^k(t)$ を車両 kが h 番目に処理する客の時間枠に対するペナルティ関数, τ_h^kを h 番目の客から $h+1$ 番目の客までの移動時間と h 番目の客のサービス時間の和とする.

このとき $f_h^k(t)$ は次の漸化式により計算される.

$$f_0^k(t) = \begin{cases} +\infty, & t \in (-\infty, e_0) \\ 0, & t \in [e_0, +\infty) \end{cases}$$

$$f_h^k(t) = \min_{t' \le t} \left(f_{h-1}^k(t' - \tau_{h-1}^k) + p_h^k(t') \right), \qquad 1 \le h \le n_k + 1. \tag{5}$$

$f_h^k(t)$ の計算には $f_{h-1}^k(t)$ を用いるが, このとき $f_{h-1}^k(t)$ を右へτ_{h-1}^kシフトしていることに注意する. ルート全体のペナルティの最小値は $\min_t f_{n_k+1}^k(t)$で求まる. 各客の最適サービス時刻 $s_{\sigma^k(h)}, h = 1, ..., n_k$と車両 kがデポに帰還する時刻 s^kは上記の漸化式 (5) で求めた $f_h^k(t), h =$

$1, ..., n_k+1$ を用いて次の漸化式より計算される．なお，漸化式中では，$s_0 = s^k$と解釈する．

$$s^k = \min \arg \min_t f^k_{n_k+1}(t)$$
$$s_{\sigma^k(h)} = \min \arg \min_{t \leq s_{\sigma^k(h+1)} - \tau^k_h} f^k_h(t), \quad 1 \leq h \leq n_k. \tag{6}$$

3.2.2 アルゴリズムの実現と時間量の評価

漸化式 (5) を計算するためのデータ構造について考える．$p^k_h(t)$ が区分線型関数であることから，$f^k_h(t)$ も区分線型関数となる．よって，これらの関数を，関数の各区分を要素とする連結リストで記憶する．すなわち，リストの要素には区分線型関数の各区間とその区間での関数が記憶される．$f^k_h(t)$ の計算では，まず，$f^k_{h-1}(t)$ の各区間をτ^k_{h-1}シフトし，$f^k_{h-1}(t-\tau^k_{h-1})$ を求める．次に，$f^k_{h-1}(t-\tau^k_{h-1})$ と $p^k_h(t)$ を，区間の区切りをマージしつつ，各区間に対応する線型関数を足し合わせることで，$f^k_{h-1}(t-\tau^k_{h-1}) + p^k_h(t)$ を求め，新たな連結リストに保持する．最後に，得られたリストを t の小さい方から順に走査しつつ，最小値の計算を行う．

関数 $f^k_{h-1}(t)$ と関数 $p^k_h(t)$ それぞれの区間の最大数は $O(\delta_k)$ であるので，$f^k_{h-1}(t)$ と $p^k_h(t)$ から $f^k_{h-1}(t-\tau_{h-1}) + p^k_h(t)$ と $f^k_h(t)$ を求める計算は $O(\delta_k)$ 時間で可能である．ペナルティの総和の計算 (3.2.1) と各客のサービス時刻の決定の計算 (6) は，ともに $f^k_h(t)$ を走査するだけなので，$O(\delta_k)$ 時間で計算できる．以上より，ルートのペナルティの総和の最小値と各客の最適サービス時刻は，全体で $O(n_k\delta_k)$ 時間で計算できる．

3.3 局所探索法の高速化

この節では，探索を高速化するための方法を 2 つ提案する．

まず，1 つ目は解の改善が見込まれない近傍を探索しないようにすることである．そのために，$m \times m$ 行列 $(c_{k,k'})$ と $(o_{k,k'})$，要素数 m の配列 Iを用意する．行列 $(c_{k,k'})$ と $(o_{k,k'})$ の各行，各列はルートに対応し，(k,k') 要素には，それぞれ，クロス交換操作，2-opt*操作によるルート$\sigma^k, \sigma^{k'}$ 間での解の改善に関する情報を保持する．また，配列 Iの第 k要素には，ルート内挿入操作によるルートσ^kでの解の改善に関する情報を保持する．すなわち，$c_{k,k'}$については，前回 $N^{cross}(\boldsymbol{\sigma}, k, k')$ に改善解が存在しないと判定された以降の探索において，ルートσ^kまたは$\sigma^{k'}$のいずれかに変更が加えられた場合には，$c_{k,k'} = 1$ とし，それ以外では $c_{k,k'} = 0$ と定義する．$(o_{k,k'})$ と配列 Iの各要素も同様に定義する．そして，$N^{cross}(\boldsymbol{\sigma}, k, k')$ の探索を $c_{k,k'} = 1$ のときに限り行うものとする．N^{2opt^*}, N^{intra}についても同様である．こうすることにより，改善の可能性のない近傍解探索を省くことができる．

もう一方の方法は，解の評価法を工夫することにより，時間ペナルティの評価を高速化できるというものである．現在の解を$\boldsymbol{\sigma}^{curr}$，近傍 $NB(\boldsymbol{\sigma}^{curr})$ 内のある解を$\boldsymbol{\sigma}^{new}$とすると，解の改善が起こるのは，以下で定義する$\Delta cost$ が正となるときである．

$$\begin{aligned} \Delta cost &= cost(\boldsymbol{\sigma}^{curr}) - cost(\boldsymbol{\sigma}^{new}) \\ &= \Delta D + \Delta T + \alpha \Delta Q \end{aligned} \tag{7}$$

ここでΔDは解$\boldsymbol{\sigma}^{curr}$ の距離の総和と$\boldsymbol{\sigma}^{new}$の距離の総和の差，$\Delta T$は解$\boldsymbol{\sigma}^{curr}$ の時間ペナルティの総和と$\boldsymbol{\sigma}^{new}$の時間ペナルティの総和の差，$\Delta Q$ は解$\boldsymbol{\sigma}^{curr}$ の容量超過量の総和と$\boldsymbol{\sigma}^{new}$の容量超過量の総和の差である．以下の節では，これらΔD, ΔT, およびΔQ の計算に要する時間について考察する．

3.3.1 ΔDについて

$$\Delta D = (\text{取り除かれる枝の距離の和}) - (\text{挿入される枝の距離の和})$$

によってΔDは計算される. クロス交換近傍と 2-opt*近傍においては, 取り除かれる枝の本数と挿入される枝の本数は定数であるので, ΔDの評価は定数時間で可能である. ルート内挿入近傍においては, 反転の操作のときに, 反転されるパス上の全ての枝について距離を再計算する必要があるので, ΔDの評価は $O(L^{intra})$ となる.

3.3.2 ΔTについて

ΔTは, 近傍操作の対象となる 2 つ (ルート内挿入操作の場合は 1 つのみ) の解に対するペナルティの変化量のみを計算すれば求まる. しかし, この計算を, 3.2節で述べた前向きペナルティ最小関数の漸化式 (5) を毎回計算し直していたのでは効率が悪い. そこで, 以下に述べる関数 $b_h^k(t)$ をあらかじめ用意しておくことで, この部分の計算の高速化を図る.

関数 $b_h^k(t)$ を, ルートσ^kにおける h 番目の客のサービス開始時刻が t 以後で, 客$\sigma^k(h), \sigma^k(h+1), ..., \sigma^k(n_k)$ をサービスするときのペナルティの最小値と定義し, これを以後, 後ろ向きペナルティ最小関数と呼ぶ. すると, $b_h^k(t)$ は, $f_h^k(t)$ の計算と対称的に, 以下の漸化式より計算できる.

$$\begin{aligned} b_{n_k+1}^k(t) &= \min_{t \le t'} p_0^k(t') \\ b_h^k(t) &= \min_{t \le t' <} \left(b_{h+1}^k(t' + \tau_h^k) + p_h^k(t') \right), \quad 1 \le h \le n_k \end{aligned} \tag{8}$$

ある h に対し, 前向きペナルティ最小関数 $f_h^k(t)$ と後ろ向きペナルティ最小関数 b_{h+1}^kが既知であれば, これらを利用することにより, ルートσ^kのペナルティの最小値 $\min_t f_{n_k+1}^k(t)$ は,

$$\min_t \left(f_h^k(t) + b_{h+1}^k(t + \tau_h^k) \right) \tag{9}$$

と計算できる. この計算は $O(\delta_k)$ で可能であるため, 漸化式 (5) を再計算するよりも効率的である. 局所探索法の近傍内の探索順序を工夫することで, 各ΔTの計算にこのアイディアを利用できる. なお, この場合, 各客に対する前向きと後ろ向きのペナルティ最小関数をあらかじめ計算して記憶しておき, さらに, 解の更新時には, 変更の加わったルートに含まれる全ての客に対してこれらの関数を計算し直す必要があるが, 通常, 解の評価回数に比べて解の更新回数は十分小さいため, 問題はないと考えられる. 以下では, 各近傍についてΔTの計算法の詳細を述べる. なお, $\tau(h, h')$ を客 h から客 h'への移動時間と客 i でのサービス時間との和, すなわち, $\tau(h, h') = u_h + t_{h,h'}$とする.

- クロス交換近傍における高速化

ルートσ^kと$\sigma^{k'}$に対するクロス交換操作を考える. ここで, $\sigma^k(i) \to \sigma^k(j), i \ge j$は客$\sigma^k(i)$から$\sigma^k(j)$ までのパスを記すものとする. なお, $i < j$のときは空パスを示すものとする. さて, $N^{cross}(\boldsymbol{\sigma}, k, k')$ の解を, 以下の 4 重ループに示す順序で探索する.

for $i_1 = 0, 1, \cdots, n_k$ **do**
  for $i_2 = 0, 1, \cdots, n_{k'}$ **do**
    for $j_1 = i_1, i_1 + 1, \cdots, i_1 + L^{cross}$ **do**
      for $j_2 = i_2, i_2 + 1, \cdots, i_2 + L^{cross}$ **do**

{ パス$\sigma^k(i_1+1) \to \sigma^k(j_1)$ とパス$\sigma^{k'}(i_2+1) \to \sigma^{k'}(j_2)$ を交換して得られる
新しい解の評価を行う }
end for
end for
end for
end for

σ^kのパス$\sigma^k(i_1+1) \to \sigma^k(j_1)$と$\sigma^{k'}$のパス$\sigma^{k'}(i_2+1) \to \sigma^{k'}(j_2)$を交換するときのコストを考える．まず，クロス交換操作によって得られる新しい2本のルートの内，$\langle$デポ $\to \sigma^k(i_1), \sigma^{k'}(i_2+1) \to \sigma^{k'}(j_2), \sigma^k(j_1+1) \to$ デポ $\rangle$ というルートについて考える．このルートにおける客$\sigma^{k'}(j_2)$までの前向きペナルティ最小関数を$f_{j_2}(t)$と記すことにする．$f_{j_2}(t)$が既知であれば，$b^k_{j_1+1}$は記憶されているので，このルートに対する時間枠ペナルティの最小値は，

$$\min_t \left[f_{j_2}\left(t - \tau(\sigma^{k'}(j_2), \sigma^k(j_1+1))\right) + b^k_{j_1+1}(t) \right] \tag{10}$$

と計算できる．さて，$f_{j_2}(t)$の計算であるが，上記のループではj_2はまずi_2に設定される．このとき，ルート$\sigma^{k'}$からは，空のパスが抜かれると解釈できるので，$f_{j_2}(t) = f^k_{i_1}(t)$となる．これ以降は，$j_2$の増加とともにデポから$j_2$までの客数が1ずつ増えていくので，$f_{j_2}(t)$は式(5)と同様の漸化式により，$O(\delta_k + \delta_{k'})$時間で計算できる．もう一方の新しいルートについても同様である．

式(10)の計算は，2つの関数の和をとり，和をとった関数をスキャンするだけなので，$O(\delta_k + \delta_{k'})$時間で可能である．以上より，クロス交換操作におけるΔTの計算量は，$O(\delta_k + \delta_{k'})$時間となる．

• 2-opt*交換近傍における高速化

ルートσ^kと$\sigma^{k'}$に対する2-opt*交換操作を考える．このとき，交換の対象となる枝が$(\sigma^k(i_1), \sigma^k(i_1+1))$と$(\sigma^{k'}(i_2), \sigma^{k'}(i_2+1))$であるとする．まず，得られる新しい2本のルートの内，ルート$\langle$デポ $\to \sigma^k(i_1), \sigma^{k'}(i_2+1) \to$ デポ $\rangle$について考える．この場合，$f^k_{i_1}(t)$と$b^{k'}_{i_2+1}(t)$はすでに記憶されているので，時間枠ペナルティの最小値は，(9)式と同様に

$$\min_t \left[f^k_{i_1}\left(t - \tau(\sigma^k(i_1), \sigma^{k'}(j_2+1))\right) + b^{k'}_{i_2+1}(t) \right] \tag{11}$$

と計算できる．この計算は前述の式(10)と同様に$O(\delta_k, \delta_{k'})$時間で可能である．

• ルート内挿入近傍における高速化

ルートσ^kに対するルート内挿入操作を考える．このとき，近傍$N^{intra}(\boldsymbol{\sigma}, k)$部分集合は，以下の2組の3重ループに示す順序により生成される．

line 1 **for** $r = 1, \cdots, L^{intra}$(取り除くパスの長さ) **do**
line 2 **for** $i = 0, 1, \cdots, n_k$ **do**
line 3 **for** $j = i, i+r+1, i+r+2, \cdots, n_k$ **do**
line 4 { ルート $\langle$デポ $\to \sigma^k(i), \sigma^k(i+r+1) \to \sigma^k(j), \sigma^k(i+1) \to \sigma^k(i+r), \sigma^k(j+1) \to$ デポ $\rangle$
line 5 の評価を行う (通常挿入の場合)}
line 6 { ルート $\langle$デポ $\to \sigma^k(i), \sigma^k(i+r+1) \to \sigma^k(j), \sigma^k(i+r) \to \sigma^k(i+1), \sigma^k(j+1) \to$ デポ $\rangle$ }
line 7 の評価を行う (反転挿入の場合)}
line 8 **end for**
line 9
line10 **for** $j = i-1, i-2, \cdots, 0$ **do**
line11 { ルート $\langle$デポ $\to \sigma^k(j), \sigma^k(i+1) \to \sigma^k(i+r), \sigma^k(j+1) \to \sigma^k(i), \sigma^k(i+r+1) \to$ デポ $\rangle$

line12　　の評価を行う (通常挿入の場合)}
line13　　{ルート $\langle$ デポ $\to \sigma^k(i), \sigma^k(i+r) \to \sigma^k(i+1), \sigma^k(j+1) \to \sigma^k(i), \sigma^k(i+r+1) \to$ デポ $\rangle$ }
line14　　の評価を行う (反転挿入の場合)}
line15　**end for**
line16　**end for**
line17　**end for**

新しく生成されるルートにおける客$\sigma^k(i)$ の前向きペナルティ最小関数を $f_i(t)$ と表すことにする. まず, line 3 で始まる **for** ループについて考える. 通常挿入の場合, $f_j(t)$ が既知であれば, 挿入されたパス$\sigma^k(i+1) \to \sigma^k(i+r)$ 上の全ての客に対して, $i+1, i+2, \cdots$の順に漸化式 (5) の計算を行えば $f_{i+r}(t)$ が求まり, これと記憶されている $b^k_{j+1}(t)$ を用いて, 新たなルートに対する時間枠ペナルティの最小値は,

$$\min_t \left[f_{i+r}\left(t - \tau(\sigma^k(i+r), \sigma^k(j+1))\right) + b^k_{j+1}(t) \right] \tag{12}$$

と計算できる. また, 反転挿入のときも同様である. これらの計算は, 漸化式 (5) の計算を $O(r)$ 回行うことでできるので, $O(r\delta_k)$ 時間で可能である.

さて, $f_j(t)$ の計算であるが, 上記のループでは jはまず i に設定されるので, $f_j(t) = f^k_i(t)$ となる. これ以降は, jはまず $i+r+1$ に設定され, さらに 1 つずつ増加していくが, それとともにパス $\langle$ デポ $\to \sigma^k(i), \sigma^k(i+r+1) \to \sigma^k(j) \rangle$ 内の客数が 1 ずつ増えていくので, $f_j(t)$ は式 (5) と同様の漸化式により各 jに対して $O(\delta_k)$ 時間で計算できる.

なお, line 3 で始まる **for** ループでは, 取り除いたパスを挿入する位置が, ルートの進行方向において取り除いた位置よりも後方に限定されている (すなわち $l_1 < i_1$の場合は考えていない). そこで, 前方への挿入も考慮するために, line10 で始まる **for** ループが必要となる. このループでのペナルティ計算は, line 3 で始まる **for** ループに対する上述のアルゴリズムを, 前向きペナルティ最小関数と後ろ向きペナルティ最小関数の役割を入れ替えて, 対称的に行うことで, 同様の計算時間で可能である.

以上の考察と, $r \leq L^{intra}$であることから, 一回あたりの $T(\sigma^k)$ の計算は, $O(\delta^k L^{intra})$ 時間で可能であることがいえた.

3.3.3　ΔQ について

各ルートに対し, ルート内の客の総要求量をあらかじめ計算し, 記憶しておく. 近傍操作の前後でのこの値の変化を考える. ルート内挿入近傍では, この値は変化しない. クロス交換近傍と 2-opt*近傍では, 交換操作の対象となる 2 つのルートの値が変化する. 以下では, この変化量の効率的計算法を示す.

ルートσ^k内の i 番目の客のサービス要求量を q^k_iと表す. このとき, ルートσ^k内の各客 i に対して, 累積要求量を,

$$\begin{aligned} &c^k_0 = 0 \\ &c^k_i = \textstyle\sum_{p=1}^{i} q^k_p = c^k_{i-1} + q^k_i, \qquad\qquad i = 1, ..., n_k \end{aligned} \tag{13}$$

と定義し, 記憶しておく. この計算は, ルートσ^kに対して全体で $O(n^k)$ で計算できる. すると, ルート内の i 番目から j番目の客による部分パスの総要求量は, $c^k_j - c^k_{i-1}$となり, 定数時間で計算できる. 局所探索を行うときの初期解の設定時と, 解の更新が行われたときに, 新しく生成されたルートに対して, ルート内の客の総要求量と c^k_iを再計算する必要があるが, これは生成されたルートの長さの時間で可能であり, それ以外のΔQ の計算は定数時間で可能となる.

3.4 暫定解について

この節では, 最良解の定義とその更新について述べる. 本研究の定式化では, 容量制約と時間枠制約を考慮制約として扱っているため, 局所最適解が, 容量制約と時間枠制約を絶対制約として扱うときの実行不可能解となる場合がある. しかし, 探索の過程で評価する解の中には, 絶対制約を全て満たす解が存在する場合がある. 応用によっては, ペナルティ付き目的関数に対する局所最適解よりも, そのような解の方が重要である場合もあると考えられるので, 探索中に保持する暫定解を更新する基準として, 目的関数 (1) とは別の評価基準を設けることを許す (もちろん目的関数 (1) をそのまま用いることも可能である). この基準となる評価関数を $best\_eval(\boldsymbol{\sigma})$ と記す.

4 反復局所探索法

この節では, メタヒューリスティクス (メタ解法, メタ戦略) の一種である反復局所探索法 (*Iterated Local Search*, *ILS*) の説明をする.

反復局所探索法とは, 局所探索法を複数回適用し, 全ての探索の中で最も良い解を出力するという方法であるが, 各反復の初期解の生成においてそれまでの探索で得た情報を用いるところにその特徴がある. その概略は下記のようなものである.

ITERATED LOCAL SEARCH (ILS)

1. 初期解$\boldsymbol{\sigma}$を生成し, $\boldsymbol{\sigma}^{seed} := \boldsymbol{\sigma}$, $\boldsymbol{\sigma}^{best} := \boldsymbol{\sigma}$とする.
2. 局所探索法により$\boldsymbol{\sigma}$を改善する. すなわち, $\boldsymbol{\sigma} := LS(\boldsymbol{\sigma})$.
3. $best\_eval(\boldsymbol{\sigma}) < best\_eval(\boldsymbol{\sigma}^{best})$ ならば, $\boldsymbol{\sigma}^{best} := \boldsymbol{\sigma}$とする.
4. $cost(\boldsymbol{\sigma}) < cost(\boldsymbol{\sigma}^{seed})$ ならば, $\boldsymbol{\sigma}^{seed} := \boldsymbol{\sigma}$とする.
5. 停止条件が満たされれば$\boldsymbol{\sigma}^{best}$を出力して終了. そうでなければ$\boldsymbol{\sigma}^{seed}$に少しだけ変形を加えて, 新たな初期解$\boldsymbol{\sigma}$を生成し, 2に戻る.

ステップ 5における$\boldsymbol{\sigma}^{seed}$の変形は, 解の構造があまり変化しないような操作を利用してしまうと, 次の局所探索において同じ局所最適解が出力される可能性が高いので, そのようなことが出来るだけ起こらないように変形を加えなければならない. 本研究では, ランダムなクロス交換操作を用いた. ランダムなクロス交換操作とは, 異なる 2 本のルートをランダムに選択し, さらにそれらの中から交換されるパスの位置と長さをそれぞれランダムに選択し, 交換を行うというものである. そして, この操作を$\boldsymbol{\sigma}^{seed}$に対して r回 (rはパラメータ) 繰り返して, 新しい初期解$\boldsymbol{\sigma}$を得る. 本研究の局所探索法では, 初期解に対して, まず, ルート内挿入近傍により, 同一ルート内の改善を行うので, 解の構造が前回の局所最適解と異なるようになり, 新しい探索によって得られる局所最適解が, 以前のものと一致することが少なくなると考えられる.

5 計算実験

実験は, ワークステーション Sun Ultra 2 Model 2300 (300 MHz, 1 GB memory) 上で C 言語を用いて行った.

5.1 問題例

用いた問題例は, Solomon のベンチマーク問題 (http://dmawww.epfl.ch/ rochat/rochat_data/solomon.html) [3] である. この問題は, $[0,100]^2$の正方形内に 100 人の客が分布しており, 各客間の距離は, ユークリッド距離で与えられる. 移動時間は距離に等しい. 各客 i には, 時間枠 $[e_i, l_i]$, 要求量 q_i, サービス時間 u_iが与えられている. 各車両の容量制約は全て等しく, その値としてある一定値が与えられている. このベンチマーク問題では, 容量制約と時間枠制約を絶対制約として扱う. これらの問題は, 客の分布と時間枠の与え方により, いくつかのタイプに分類されている. 客の分布については, タイプ R, C, および RC の 3 つ, 時間枠については, タイプ 1 と 2 の 2 つがある. これら全ての組合せを考え, R1, C1, RC1, R2, C2, RC2 の 6 タイプが与えられている. R タイプでは, 客が一様に分布しており, C タイプでは, 10 人を 1 グループとする 10 グループが, それぞれ固まって分布している. RC タイプは, 両方のタイプの性質をおり混ぜたものとなっている. また, タイプ 1 では, デポの時間枠が短く, そのため各客の時間枠を満たしてサービスを行う場合, 比較的多くの車両が必要となる. それに対してタイプ 2 では, デポの時間枠が長く, 少数の車両で全ての客の時間枠を満たしながら, サービスを行うことが可能である.

容量制約と時間枠制約を絶対制約として扱うことから, 3.4節で述べた $best\_eval(\boldsymbol{\sigma})$ としては, $Q(\boldsymbol{\sigma}), T(\boldsymbol{\sigma}), D(\boldsymbol{\sigma})$ の辞書式順に最小化を行うという評価基準を用いた. 本研究のアルゴリズムの評価は, この評価基準で得られる暫定解の精度で測るものとする. また, 各客 i のサービス開始時刻に対するペナルティ関数としては,

$$p_i(t) = \begin{cases} w(e_i - t), & t < e_i \\ 0, & e_i \leq t \leq l_i \\ w(t - l_i), & l_i < t \end{cases} \tag{14}$$

を用いることにする. ここで, wはパラメータである. さらに, 本実験においては, L^{cross} と L^{intra} の値は等しいものとし, これ以降 L と表記するものとする.

5.2 本研究のアルゴリズムの性能評価

各問題例に対してパラメータを設定して反復局所探索法を適用し, 文献 [4] に掲載されているこのベンチマーク問題に対して出されている最良解との比較を行う. なお, 車両台数は, 文献 [4] で報告されている解に用いられた台数とした. 反復局所探索の停止条件は以下のように定める.

- タイプ 1 の問題例に対しては, 1800 秒で探索を打ち切る.
- タイプ 2 の問題例に対しては, 3600 秒で探索を打ち切る.

ただし, 停止時刻において実行されている局所探索法は終了せず, 局所最適解が求まるまで探索を続ける. 結果を表 1 と表 2 に示す. これらの表において, Dは各ルートの総距離の和を表し, Tは時間枠からのサービス開始時刻のずれの総和を表している. 全ての解において容量制約は満たされているので, Q の値は省略した. よって, $T = 0$ の場合は, 容量制約と時間枠制約を絶対制約とみなしたときの実行可能解となっている. なお, 最も右の列のおける*は, 本研究の結果が, 現在までに知られている最良解と同等の解を出したことを示しており, また, **はそれを更新したことを示している. これより, タイプ Cの問題例に対しては, ほとんどの問題例について, 等しい精度の解を得ていることが分かる. また, 他のほとんどの問題例についても, ほぼ同程度の精度の解を得ていることが観測できる.

表 1: これまでの最良解と本研究の最良解の比較 (1)

問題タイプ	車両台数	†これまでの最良解		本研究の最良解		
		D	T	D	T	
r101	18	1607.7	0	1636.28	0.30	
r102	17	1434.0	0	1498.43	0	
r103	13	1207	0	1183.20	4.24	
r104	10	982.01	0	1012.69	0	
r105	14	1377.11	0	1385.19	0	
r106	12	1252.03	0	1263.61	0	
r107	10	1126.69	0	1133.70	0	
r108	9	968.59	0	1026.53	0.55	
c101	10	828.94	0	828.94	0	*
c102	10	828.94	0	828.94	0	*
c103	10	828.06	0	828.06	0	*
c104	10	824.78	0	824.78	0	*
c105	10	828.94	0	828.94	0	*
c106	10	828.94	0	828.94	0	*
c107	10	828.94	0	828.06	0	*
c108	10	828.94	0	824.78	0	*
rc101	14	1669	0	1728.21	1.81	
rc102	12	1554.75	0	1585.07	5.46	
rc103	11	1110	0	1299.57	0	
rc104	10	1135.83	0	1190.41	0	
rc105	13	1643.38	0	1572.42	2.95	
rc106	11	1448.26	0	1458.72	3.35	
rc107	11	1230.54	0	1268.95	0	
rc108	10	1139.82	0	1268.43	2.13	

†文献 [4] に掲載されている最良解
*同精度の解

6 まとめ

時間枠制約付き配送計画問題の解法の一つとして, 動的計画法を用いて各客の最適サービス時刻を決定することを局所探索法に組み込んだ解法を提案した. 代表的なベンチマーク問題に対する計算実験の結果より, 時間枠制約を考慮制約にした本研究のアルゴリズムは, 汎用的な解法であるにもかかわらず, 従来の方法に匹敵する性能を持つことが確認できた.

今後の課題として, 近傍の縮小やより効果的な近傍を考えることにより, 局所探索法 1 回あたりに要する時間の短縮が挙げられる. また, タブー探索法などの他のメタヒューリスティクスと組合せることなども考えている.

表 2: これまでの最良解と本研究の最良解の比較 (2)

問題タイプ	車両台数	†これまでの最良解		本研究の最良解		
		D	T	D	T	
r201	4	1254.80	0	1265.78	0	
r202	3	1214.28	0	1211.75	0	**
r203	3	948.74	0	963.05	0	
r204	2	869.29	0	902.68	0	
r205	3	1038.72	0	1059.98	0	
r206	3	833	0	937.95	0	
r207	3	814.78	0	839.74	0	
r208	2	738.60	0	771.65	0	
c201	3	591.56	0	591.56	0	*
c202	3	591.56	0	591.56	0	*
c203	3	591.17	0	591.17	0	*
c204	3	590.60	0	604.28	0	
c205	3	588.88	0	588.88	0	*
c206	3	588.49	0	588.49	0	*
c207	3	588.29	0	588.29	0	*
c208	3	588.32	0	588.32	0	*
rc201	4	1294	0	1498.57	0	
rc202	4	1164.25	0	1161.79	0	**
rc203	3	1079.57	0	1097.27	0	
rc204	3	806.75	0	821.54	0	
rc205	4	1328.21	0	1330.20	0	
rc206	3	1158.81	0	1207.64	0	
rc207	3	1082.32	0	1118.72	0	
rc208	3	833.97	0	829.69	0	**

†文献 [4] に掲載されている最良解
**更新解　*同精度の解

参考文献

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability,* Freeman (1979).

[2] J. Y. Potvin, T. Kervahut, B. L. Garcia and J. M. Rousseau, "The Vehicle Routing Problem with Time Windows. Part 1: Tabu Search," *INFORMS Journal on Computing,* Vol.8, No.2, 158–164 (1996).

[3] M. M. Solomon, "The Vehicle Routing and Scheduling Problems with Time Window Constraints," *Operations Research,* Vol.35, No.2, 254–265 (1987).

[4] E. Taillard, P. Badeau, M. Gendreau, F. Guertin and J. Y. Potvin, "A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows," *Transportation Science,* Vol.31, No.2, 170–186 (1997).